UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA E MECATRÔNICA

***Disciplina*: GEM03 (Algoritmos e Programação de Computadores)**
***Prof*: Ezequiel Roberto Zorzal**

Introdução a Lógica de Programação

Podemos relacionar lógica com a "correção do pensamento", pois uma de suas preocupações é determinar quais operações são válidas e quais não são, fazendo análises das formas e leis do pensamento.
Lógica: "é a arte de bem pensar"
"Ciência das formas do pensamento".

Lógica do dia-a-dia:
a)
1. A gaveta está fechada;
2. A caneta está dentro da gaveta;
3. Precisamos primeiro abrir a gaveta para depois pegar a caneta.

b)
1. Fulano é engenheiro;
2. Todo engenheiro é estudioso;
3. Logo, Fulano é estudioso.

c)
1. Anacleto é mais velho que Felisberto;
2. Felisberto é mais velho que Marivaldo;
3. Portanto, Anacleto é mais velho que Marivaldo.

F ou V = 0 ou 1

Problemas de lógica
Exemplo 1

1. Lisa tem 13 anos;
2. Virgínia não tem 12 anos;
3. A criança que comeu uma maçã não tem 11 anos;
4. Lisa não comeu nem o biscoito nem a maça.

		Lanche			Idade		
		Biscoito	Chocolate	Maçã	11 anos	12 anos	13 anos
Nome	Alexandre	0	0	1	0	1	0
	Lisa	0	1	0	0	0	1
	Virgínia	1	0	0	1	0	0
Idade	11 anos	1	0	0			
	12 anos	0	0	1			
	13 anos	0	1	0			

Nome	Lanche	Idade
Alexandre	Maçã	12 anos
Lisa	Chocolate	13 anos
Virgínia	Biscoito	11 anos

Exemplo 2.
1. Marli tomou chololate quente;
2. A mulher que comprou um casaco de lã, tomou uma sopa;
3. Regina comprou um casaco de camurça.

		Bebida			Casaco		
		Chá	Chocolate Quente	Sopa	Camurça	Couro	Lã
Nome	Marli	0	1	0	0	1	0
	Paula	0	0	1	0	0	1
	Regina	1	0	0	1	0	0
Casaco	Camurça	1	0	0			
	Couro	0	1	0			
	Lã	0	0	1			

Nome	Bebida	Casaco
Marli	Chocolate Quente	Couro
Paula	Sopa	Lã
Regina	Chá	Camurça

Malba Tahan – O homem que calculava

Haviam 5 escravas de um poderoso califa. Três delas tem olhos azuis e nunca falam a verdade. As outras duas tem olhos negros e só dizem verdade. As escravas se apresentaram com os rostos cobertos por véus e Beremiz foi desafiado a determinar a cor dos olhos de cada uma, tendo o direito a fazer três perguntas, não mais do que uma pergunta a cada escrava. Para facilita as referencias, chamaremos as 5 escravas A,B,C,D e E.
Beremiz começou perguntando à escrava A: "Qual a cor dos seus olhos?" Para seu desespero, ela respondeu em chinês, língua que ele não entendia, por isso protestou. Seu protesto não foi aceito, mas ficou decidido que as respostas seguintes seriam em árabe. Em seguida, lê perguntou a B: "Qual foi a resposta que A me deu?" B respondeu: "Que seus olhos eram azuis". Finalmente, Beremiz perguntou a C: "Quais as cores dos olhos de A e B?" A resposta de C foi: "Ä tem olhos pretos e B tem olhos azuis". Neste ponto, o homem que calculava concluiu. "A tem olhos pretos, B azuis, C pretos, D azuis e E azuis". Acertou e todos ficaram maravilhados.
Explicação para a dedução de Beremiz: Em primeiro lugar, se perguntarmos a qualquer das cinco escravas qual a cor dos seus olhos, sua resposta só poderá ser "Negros", tenha ela olhos azuis ou negros, pois na primeira hipótese ela mentirá e na segunda dirá a verdade. Logo B mentiu e, portanto seus olhos são azuis. Como C disse que os olhos de B eram azuis, C falou a verdade, logo seus olhos são negros. Também porque C fala a verdade, os olhos de A são negros, Como somente duas escravas tem olhos negros, segue-se que os olhos de D e E são azuis.

A	B	C	D	E
Negros	Azuis	Negros	Azuis	Azuis

Mas e a lógica de programação?

Significa o uso correto das leis do pensamento, da “ordem da razão” e processos de raciocínio e simbolização formais na programação de computadores, objetivando racionalidade e o desenvolvimento de técnicas que cooperem para a produção de soluções logicamente válidas e coerentes, que resolvam com qualidade os problemas que se deseja programar.
O objetivo principal do estudo da lógica de programação é a construção de algoritmos coerentes e válidos. **Afinal o que é algoritmo?**

Algoritmo:

- É uma seqüência de passos que visam atingir um objetivo bem definido.
- É uma seqüência finita e bem definida de passos que, quando executados, realizam uma tarefa específica ou resolvem um problema.

Exemplo: Receitas de culinária, manual de instruções, coreografia, etc.

Propriedades do algoritmo:

- Composto por ações simples e bem definidas (Não pode haver ambigüidade, ou seja, cada instrução representa uma ação que deve ser entendida e realizada);
- Seqüência ordenada de ações;
- Conjunto finito de passos.

Como saber se já temos detalhes suficientes para o algoritmo ser entendido e realizado?
R: Depende da relação de instruções reconhecidas pelo agente executor do algoritmo.
Exemplo:

Receita de bolo → Ser Humano;
Algoritmo Computacional → Computador.

Jogo a travessia do rio

As regras são as seguintes:
1 - Somente o pai, a mãe e o policial sabem pilotar o barco;
2 - A mãe não pode ficar sozinha com os filhos;
3 - O pai não pode ficar sozinho com as filhas;
4 - O prisioneiro não pode ficar sozinho com nenhum integrante da família;
5 - O barco só pode transportar 2 pessoas por vez.
6 - Você pode ir e vir com as pessoas quantas vezes precisar.

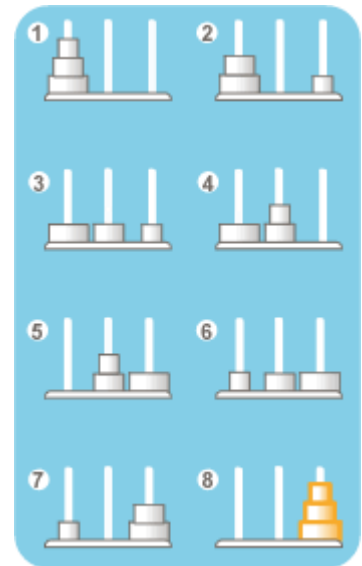
Solução:

Viagem	Ida	Volta
1	Polícia Prisioneiro	Policia
2	Policia Filha 1	Policia Prisioneiro
3	Mãe Filha 2	Mãe
4	Mãe Pai	Pai
5	Policia Prisioneiro	Mãe
6	Mãe Pai	Pai
7	Pai Filho 1	Policia Prisioneiro
8	Policia Filho 2	Policia
9	Policia Prisioneiro	-

Torre de Hanói:

Regra: Mover todos os discos de uma haste para outra sem que o disco maior fique sobre o disco menor.

Solução:

1. Posição inicial;
2. Move o disco 1 para a haste C;
3. Move o disco 2 para a haste B;
4. Move o disco 1 para a haste B;
5. Move o disco 3 para a haste C;
6. Move o disco 1 para a haste A;
7. Move o disco 2 para a haste C;
8. Move o disco 1 para a haste C;

Recipientes
Temos 3 recipientes de tamanhos distintos (8,5 e 3 litros), sendo que o recipiente de 8 litros está totalmente cheio. Considerando que os recipientes não sejam adequados, deseja-se colocar 4 litros em dois recipientes.

Solução:

Passos	Recipiente 8 litros	Recipiente 5 litros	Recipiente 3 litros
0	8	0	0
1	3	5	0
2	3	2	3
3	6	2	0
4	6	0	2
5	1	5	2
6	1	4	3
7	4	4	0

Desenvolvendo Algoritmos e Diagrama de Bloco

O objetivo principal do estudo da lógica de programação é a construção de algoritmos coerentes e válidos.
Um algoritmo pode ser definido como uma seqüência de passos que visam a atingur um objetivo bem definido.

Regras para construção de algoritmos

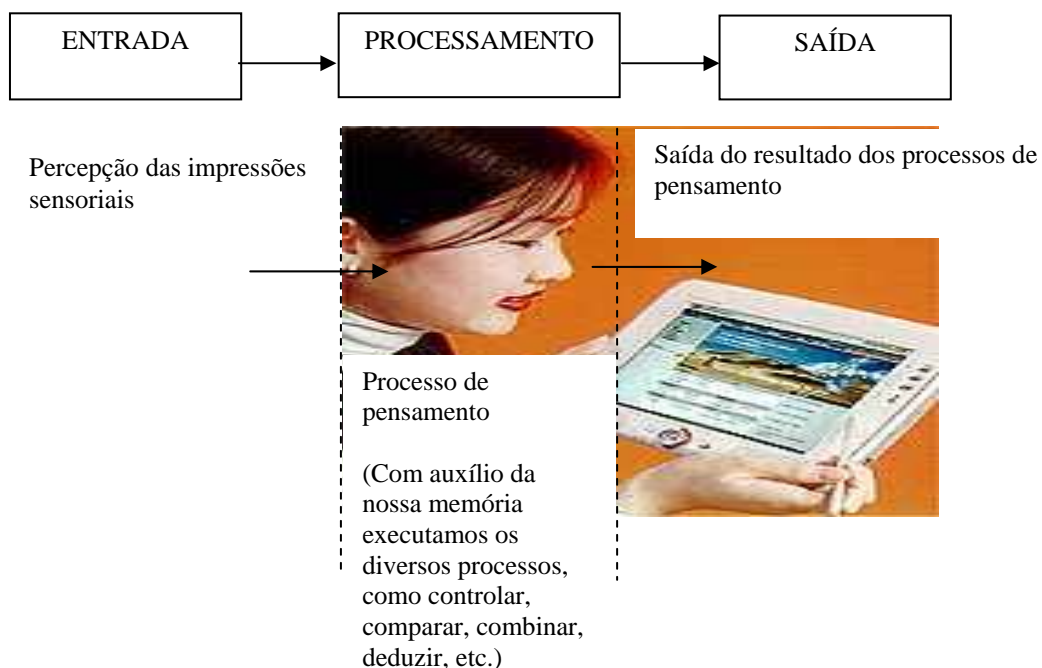
Para escrever um algoritmo precisamos descrever a seqüência de instruções, de maneira simples e objetiva. Para isso utilizaremos algumas técnicas:

- Usar somente um verbo por frase;
- Imaginar que você está desenvolvendo um algoritmo para pessoas que não trabalham com informática;
- Usar frases curtas e simples;
- Ser objetivo;
- Procurar usar palavras que não tenham sentido dúbio.

Entretanto ao montar um algoritmo precisamos primeiro dividir o problema apresentado em 3 fases fundamentais.

Entrada: São os dados de entrada do algoritmo;

Processamento: São os procedimentos utilizados para chegar ao resultado final;
Saída: São os dados já processados.

Analogia: Ser humano.

Exemplo de algoritmo:
Imagine o seguinte problema: calcular a média final dos alunos da 3ª série. Os alunos realizarão quatro provas: P1, P2, P3 e P4.
Onde:

$$MédiaFinal = \frac{P1 + P2 + P3 + P4}{4}$$

Para montar o algoritmo proposto, faremos 3 perguntas:

a) Quais são os dados de entrada?
   R: Os dados de entrada são P1, P2, P3 e P4.

b) Qual será o processamento a ser utilizado?
   R: O processamento será somar todos os dados de entrada e dividi-los por 4.

$$\frac{P1 + P2 + P3 + P4}{4}$$

c) Quais serão os dados de saída?
   R: O dado de saída será a média final.

Algoritmo: (português coloquial)
Receba a nota da Prova1
Receba a nota da Prova2
Receba a nota da Prova3
Receba a nota da Prova4
Some todas as notas e
Divida o resultado por 4
Mostre o resultado da divisão

Descrevemos então uma atividade bem cotidiana: trocar uma lâmpada.

Apesar de parecer óbvio demais, muitas vezes fazemos este tipo de atividade inconscientemente, sem percebermos os pequenos detalhes.

Algoritmo 1.0 – Troca de lâmpada.

1. Pegar uma escada;
2. Posicionar embaixo da lâmpada;
3. Buscar uma lâmpada nova;
4. Subir na escada;
5. Retirar a lâmpada velha;
6. Colocar a lâmpada nova.

Para se trocar a lâmpada, é seguida uma determinada seqüência de ações, representadas através desse algoritmo. Como isso pode ser seguido por qualquer pessoa, estabelece-se aí um padrão de comportamento.

A sequencialização tem por objetivo reger o fluxo de execução, determinando qual ação vem a seguir. O algoritmo anterior tem um objetivo bem específico: trocar uma lâmpada. E se a lâmpada não estiver queimada? O algoritmo faz com ela seja trocada do mesmo modo, não prevendo essa situação.

Para solucionar este problema, podemos efetuar um teste seletivo, verificando se a lâmpada está ou não queimada:

Algoritmo 1.1 – Troca de Lâmpada com teste.

1. Pegar uma escada
2. Posicionar embaixo da lâmpada
3. Buscar uma lâmpada nova;
4. Ligar o interruptor;
5. Se a lâmpada não acender, então:
   - Subir na escada
   - Retirar a lâmpada velha
   - Colocar a lâmpada nova.

Dessa forma, algumas ações estão ligadas à condição (lâmpada não acender). No caso da lâmpada acender, as três linhas:

1. Subir na escada;
2. Retirar a lâmpada velha
3. Colocar a lâmpada nova.

Não serão executadas, em algumas situações, embora o algoritmo resolva o problema proposto, a solução pode não ser a mais eficiente.

Exemplo: três alunos devem resolver um determinado problema:

- O aluno A conseguiu resolver o problema executando 35 linhas de programa.
- O aluno B resolveu o problema executando 10 linhas de programa.
- O aluno C resolveu o problema executando 54 linhas de programa.

Obviamente, o algoritmo desenvolvido pelo aluno B é menor e mais eficiente que os demais. Isso significa que há código desnecessário nos demais programas.

Dessa forma, podemos otimizar o algoritmo anterior, uma vez que buscamos a escada e a lâmpada sem saber se serão necessárias:

Algoritmo 1.2 – Troca de lâmpada com teste no inicio.

1. Ligar o interruptor
2. Se a lâmpada não acender, então:
   - Pegar uma escada;
   - Posicionar embaixo da lâmpada;
   - Buscar uma lâmpada nova;
   - Subir na escada;

- Retirar a lâmpada velha;
- Colocar a lâmpada nova.

Podemos considerar ainda que a lâmpada nova pode não funcionar.
Nesse caso devemos trocá-la novamente, quantas vezes for necessário, até que a lâmpada acenda:

Algoritmo 1.3 – Troca de Lâmpada com teste de repetição indefinida.

1. Ligar o interruptor
2. Se a lâmpada não acender, então :
   - Pegar uma escada.
   - Posicionar embaixo da lâmpada
   - Buscar uma lâmpada nova;
   - Subir na escada;
   - Retirar a lâmpada velha;
   - Colocar a lâmpada nova;
   - Se a lâmpada não acender, então :
     - Retirar a lâmpada;
     - Colocar outra lâmpada;
       - Se a lâmpada não acender, então: ...

Até quando?

Observamos que o teste da lâmpada nova é efetuado por um conjunto de ações:

- Se a lâmpada não acender então:
- Retire a lâmpada
- Coloque outra lâmpada

Em vez de escrevermos várias vezes este conjunto de ações, podemos alterar o fluxo seqüencial de execução do programa, de forma que, após executar a ação “ coloque outra lâmpada” , voltemos a executar a ação “ se a lâmpada não acender” .

Precisa-se então determinar um limite para tal repetição, para garantir que ela cesse quando a lâmpada finalmente acender:

1. Enquanto a lâmpada não acender, faça:
2. Retire a lâmpada
3. Coloque outra lâmpada

Uma versão final do algoritmo, que repete ações até alcançar o seu objetivo: trocar a lâmpada queimada por uma que funcione, é apresentada abaixo.

Algoritmo 1.4 – Troca de lâmpada com teste e condição de parada.

1. Ligar o interruptor
2. Se a lâmpada não acender, então:
3. Pegar uma escada
4. Posicionar embaixo da lâmpada
5. Buscar uma lâmpada nova
6. Subir na escada
7. Retirar a lâmpada velha
8. Colocar a lâmpada nova
   - Enquanto a lâmpada não acender, faça:
   - Retirar a lâmpada
   - Colocar outra lâmpada.

O que faríamos se tivéssemos mais soquetes a testar, por exemplo, 10 soquetes?
Até agora, estamos efetuando a troca de uma única lâmpada. Todo o procedimento poderia ser repetido 10 vezes, por exemplo, no caso de querermos trocar 10 lâmpadas.

Algoritmo 1.5 – Troca de Lâmpada com teste para 10 soquetes com repetição.

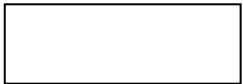
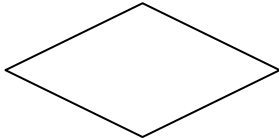
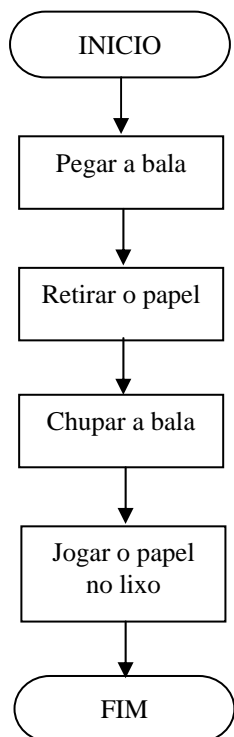
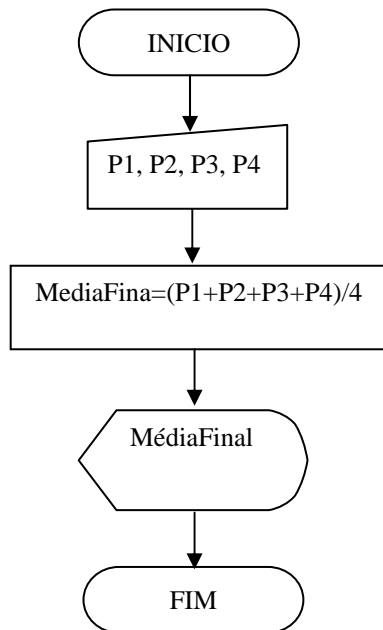
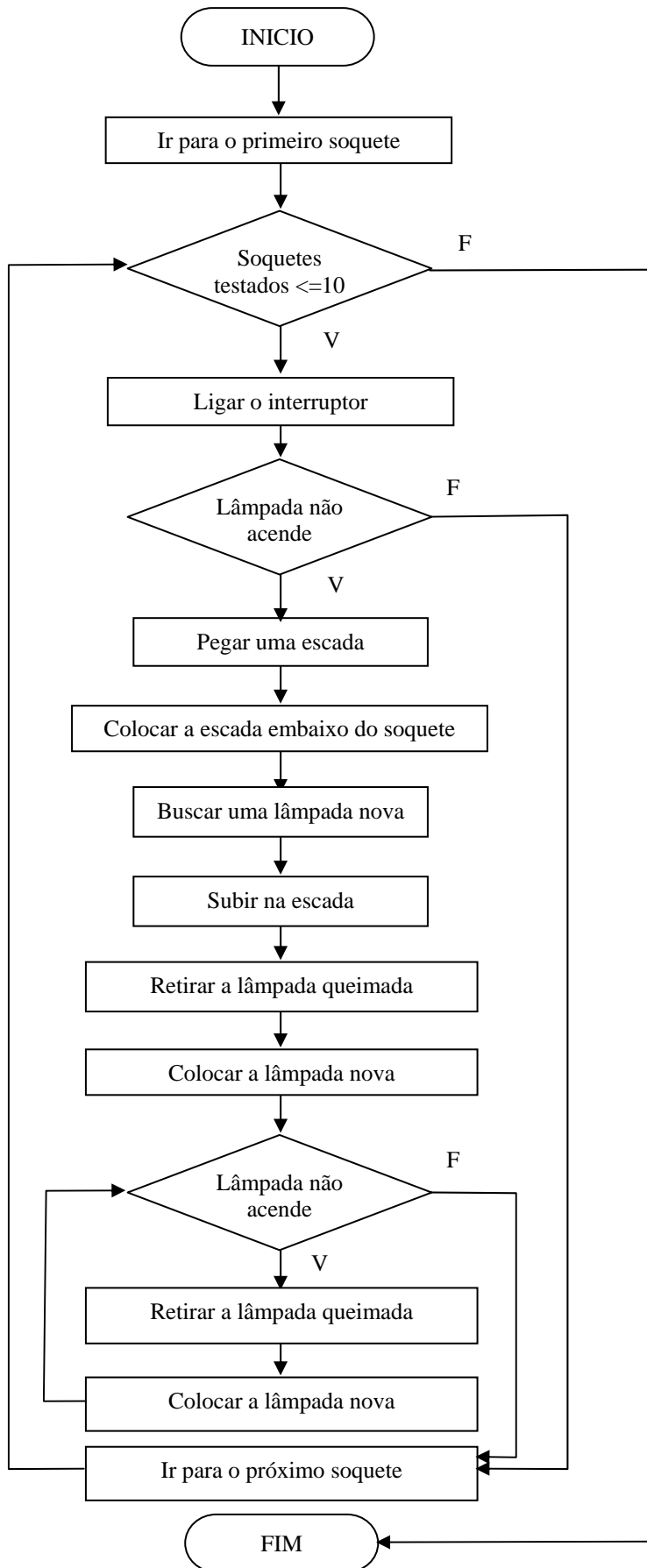
1. Ir até o interruptor do primeiro soquete;
2. Enquanto a quantidade de soquetes testados for menor que dez, faça
    - Ligar o interruptor
    - Se a lâmpada não acender, então:
    - Pegar uma escada
    - Posicionar embaixo da lâmpada
    - Buscar uma lâmpada nova
    - Subir na escada
    - Retirar a lâmpada velha
    - Colocar a lâmpada nova
        1. Enquanto a lâmpada não acender, faça:
        2. Retirar a lâmpada
        3. Colocar outra lâmpada.
3. Ir até o interruptor do próximo soquete.

De que maneira representamos o algoritmo?

Diagrama de bloco (Fluxograma)
O diagrama de blocos é uma forma padronizada e eficaz para representar os passos lógicos de um determinado processamento.

Simbologia
Símbolos mais utilizados:

Símbolo	Função
Terminal	Indica o inicio ou fim de um processamento. Exemplo: Inicio do algoritmo.
Processamento	Processamento em geral. Exemplo: Cálculo de dois números.
Entrada de dado manual	Indica entrada de dados através do teclado. Exemplo: Digite a nota da prova.
Exibir	Mostra informações ou resultados. Exemplo: Mostre o resultado do cálculo.
Condição	Indica uma condição Exemplo: Soquetes testados <=10

Exemplos:

"Chupar uma bala"

"Calcular a média de 4 notas"

Fluxograma: Algoritmo Troca de Lâmpada com teste para 10 soquetes com repetição.

Constantes, Variáveis, Tipos de Dados, Operadores e Operações Lógicas

Variáveis e constantes são os elementos básicos que um programa manipula.
Constante: é um determinado valor fixo que não se modifica ao longo do tempo, durante a execução do programa.

Variável: é a representação simbólica dos elementos de um certo conjunto. Cada variável corresponde a uma posição de memória, cujo conteúdo pode ser alterado ao longo do tempo de execução de um programa.

As variáveis podem ser:
Numéricas: inteiras, reais.
Caracteres: conjunto de caracteres, string.
Lógicas: V ou F
Alfanuméricas: Letras e/ou números.

Declaração de variáveis

As variáveis só podem armazenar valores de um mesmo tipo, de maneira que também são classificadas sendo numéricas, lógicas e literais.

Exemplos de declarações:
Inteiro: x,y;
Caracter: nome, endereço, data;
Real: abc, peso, pi;
Lógico: resposta;

Operadores

Os operadores são meios pelo qual incrementamos, decrementamos, comparamos e avaliamos dados dentro do computador. Temos 3 tipos de operadores:
Operadores Aritméticos;
Operadores relacionais;
Operadores Lógicos;

Operadores aritméticos: Utilizados para obter resultados numéricos.

Operador	**Função**	**Exemplos**
+	Adição	2+3,X+Y
-	Subtração	4-2, N-M
*	Multiplicação	3*4, A*B
/	Divisão	10/2, X1/X2
pot(x,y)	Potenciação (x elevado a y)	pot(2,3)
rad(x,y)	Radiciação (Raiz quadrada de x)	rad(9)
mod	Resto da divisão	9 mod 4 resulta em 1
div	Quociente da divisão	27 div 5 resulta em 5

Precedência entre os operadores aritméticos:

Prioridade	**Operadores**
1ª	Parênteses mais internos
2ª	pot rad
3ª	* / div mod
4ª	+ -

Em caso de empate (operadores de mesma prioridade), devemos resolver da esquerda para a direita, conforme a seqüência existente na expressão aritmética.
Exemplos:

a)
5 + 9 + 7 + 8/4
5 + 9 + 7 + 2
23

b)
1 – 4 * 3/6 – pot(3,2)
1 – 4 * 3/6 – 9
1 – 12/6 – 9
1 – 2 – 9
-10

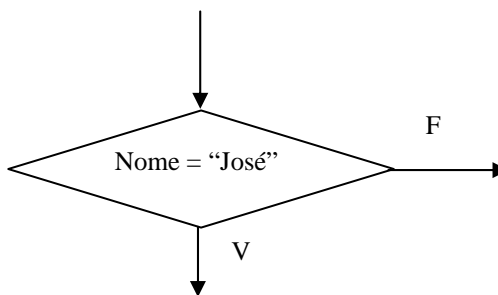
c)
pot(5,2) – 4/2 + rad(1 + 3 * 5)/2
pot(5,2) – 4/2 + rad(1 + 15)/2
pot(5,2) – 4/2 + rad(16)/2
25 – 4/2 + 4/2
25 – 2 +2
25

Operadores relacionais
Utilizados para comparar cadeias de caracteres e números. Sempre retornam valores lógicos (V ou F).

Operador	**Função**	**Exemplos**
=	Iagual a	3 = 3, X = Y
>	Maior que	5 > 4, X > Y
<	Menor que	3 < 6, X < Y
>=	Maior ou igual a	5 >= 3, X >= Y
<=	Menor ou igual a	3 <= 5, X <= Y
<>	Diferente de	8 <> 9, X <> Y

Exemplos:

Tendo duas variáveis A=5 e B=3

Expressão	**Resultado**
A = B	F
A <> B	V
A > B	V
A < B	F
A >= B	V
A <= B	F

a)
2 * 4 = 24/3
8 = 8
V
b)
15 mod 4 < 19 mod 6
3 < 1
F
c)
3 * 5 div 4 <= pot(3,2)/0,5
15 div 4 <= 9/0,5
3 <= 18
V
d)
2 + 8 mod 7 >= 3 * 6 – 15
2 + 1 >= 18 – 15
3 >= 3
V

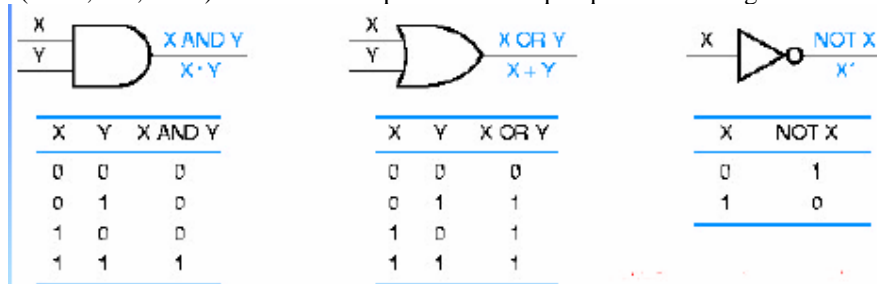
Operadores Lógicos

Operador	**Função**
Não (NOT)	Negação
e (AND)	Conjunção
ou (OR)	Disjunção

Três portas básicas (AND, OR, NOT) são suficientes para construir qualquer sistema lógico combinacional digital.

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

X	NOT X
0	1
1	0

Operação de negação

A	**não A**
F	V
V	F

Operação de conjunção

A	**B**	**A e B**
F	F	F
F	V	F
V	F	F
V	V	V

Operação de disjunção não-exclusiva

A	**B**	**A ou B**
F	F	F
F	V	V
V	F	V
V	V	V

Precedência entre os operadores lógicos

Prioridade	**Operadores**
1ª	não
2ª	e
3ª	ou

Exemplos:
a)
Se chover **e** relampejar, eu fico em casa.
Quando eu fico em casa?

b)
Se chover **ou** relampejar eu fico em casa.
Quando eu fico em casa?

c)
2 < 5 e 15/3 = 5
V e 5 = 5
V e V
V

d)
F ou 20 div(18/3) <> (21/3) div 2
F ou 20 div 6 <> 7 div 2
F ou 3 <> 3
F ou F
F

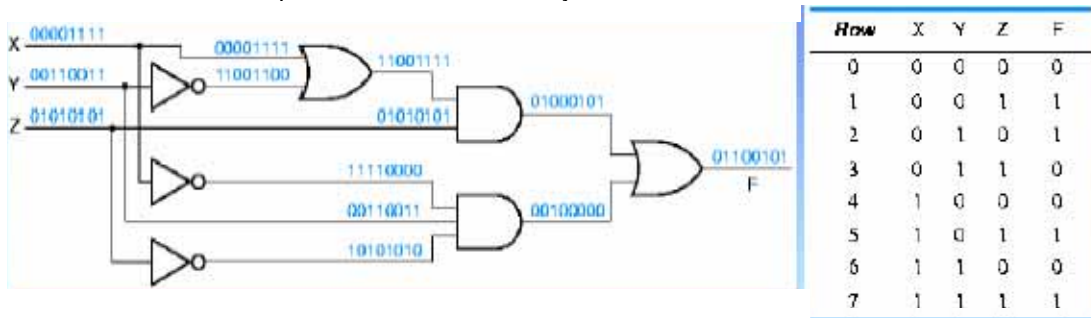
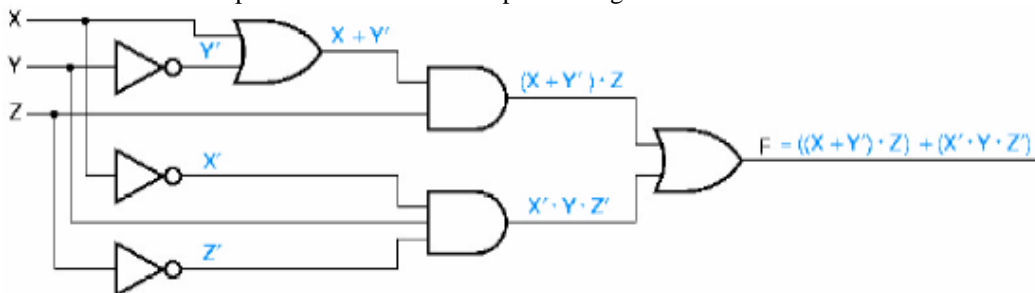
e)
não V ou pot(3,2)/3 < 15 – 35 mod 7
não V ou 9/3 < 15 – 0
não V ou 3 < 15
não V ou V
F ou V
V

f)
não (5 <> 10/2) ou V e 2 – 5 > 5 – 2 ou v)
não (5 <> 5 ou V e – 3 > 3 ou V)
não (F ou V e F ou V)
não (F ou F ou V)
não (F ou V)
não (V)
F

g)
pot(2,4) <> 4 + 2 ou 2 + 3 * 5/3 mod 5 < 0
16 <> 6 ou 2 + 15/3 mod 5 < 0
V ou 2 + 5 mod 5 < 0
V ou 2 + 0 < 0
V ou 2 < 0
V ou F
V

Exemplo:

Podemos obter a tabela-verdade a partir de todas as combinações das entradas.

Row	X	Y	Z	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

Diretamente da tabela verdade é possível escrever uma expressão lógica.

Resolução:

F ← ((X ou não(Y)) e Z) ou (não(X) e não(Y) e não(Z));

Comandos de Atribuição

Comandos de Atribuição
Um comando de atribuição permite-nos fornecer um valor a uma variável.

Exemplos:
lógico: A,B;
inteiro: X;
A ← B;
X ← 8 + 13 div 2;
B ← 5 = 3;
X ← 2;

Comandos de entrada e saída
Entrada de dados:
Para que o algoritmo possa receber os dados de que necessita, adotaremos um comando de entrada de dados denominado leia, cuja finalidade é atribuir o dado a ser fornecido à variável identificada.

leia (X);
leia (A, XPTO, NOTA);

Saída de dados:
Para que o algoritmo possa mostrar os dados que calculou, como resposta ao problema que resolveu, adotaremos um comando de saída de dados denominado escreva, cuja finalidade é exibir o conteúdo da variável identificada.

escreva (Y);
escreva (B, XPTO, SOMA/4);
escreva (“Bom dia”, NOME);
escreva (“Você pesa ”, P, “ quilos”);

Blocos
Um bloco pode ser definido como um conjunto de ações com uma função definida; nesse caso, um algoritmo pode ser visto como um bloco. Ele serve também para definir limites nos quais as variáveis declaradas em seu interior são conhecidas. Para delimitar um bloco, utilizamos os delimitadores inicio e fim.

Exemplo:
início //início do bloco (algoritmo)
// declaração de variáveis
// seqüência de ações
fim. //fim do bloco (algoritmo)

Estrutura seqüencial
A estrutura seqüencial de um algoritmo corresponde ao fato de que o conjunto de ações primitivas será executado em uma seqüência linear de cima para baixo e da esquerda para a direita, isto é, na mesma ordem em que foram escritas. Convencionaremos que as ações serão seguidas pó um ponto-e-vírgula (;), que objetiva separar uma ação da outra e auxiliar a organização seqüencial das ações, pois após encontrar um (;) deveremos executar o próximo comando da seqüência.

Modelo geral de um algoritmo:
1. início //identificação do início do bloco correspondente ao algoritmo
2. //declaração de variáveis
3. //corpo do algoritmo
4. ação 1;
5. ação 2;
6. ação 3;
7. .
8. .
9. .
10. ação n;
11. fim. //fim do algoritmo

Exemplos:

a)
Construa um algoritmo que calcule a média aritmética entre quatro notas bimestrais quaisquer fornecidas por um aluno (usuário).
Dados de entrada: quatro notas bimestrais (N1, N2, N3, N4).
Dados de saída: média aritmética anual (MA).

O que devemos fazer para transformar quatro notas bimestrais em uma média anual?
Resposta: utilizar a média aritmética.

O que é média aritmética?
Resposta: a soma dos elementos divididos pela quantidade deles. Em nosso caso particular: (N1, N2, N3, N4)/4.

Algoritmo – Média aritmética

1. início //Começo do algoritmo

```
2.    //declaração de variáveis
3.    real: N1, N2, N3, N4, MA; //Notas bimestrais e média anual
4.
5.    //entrada de dados
6.    leia(N1, N2, N3, N4);
7.
8.    //processamento
9.    MA ← (N1 + N2 + N3 + N4)/4;
10.
11.   //saída de dados
12.   escreva (MA);
13.
14. fim.    //Término do algoritmo
```

b)
Construa um algoritmo que calcule a quantidade de latas de tinta necessárias e o custo para pintar tanques cilíndricos de combustível, em que são fornecidos a altura e o raio desse cilindro.
Sabendo que:

- A lata de tinta custa R$ 50,00;
- Cada lata contém 5 litros;
- Cada litro de tinta pinta 3 metros quadrados.

Dados de entrada: altura (H) e raio (R).
Dados de saída: custo (C) e quantidade (QTDE).
Utilizando o planejamento reverso, sabemos que:

- O custo é dado pela quantidade de latas * R$ 50,00;
- A quantidade de latas é dada pela quantidade total de litros/5;
- A quantidade total de litros é dada pela área do cilindro/3;
- A área do cilindro é dada pela área da base + área lateral;
- A área da base é (PI * pot(R,2));
- A área lateral é altura * comprimento: (2 * PI * R * H);
- Sendo que R (raio) e H (altura) são dados de entrada e PI é uma constante de valor conhecido: 3,14.

```
1.  início
2.      real: H, R;
3.      real: C, Qtde, Área, Litro;
4.      leia (H, R);
5.      Área ← (3,14 * pot(R,2)) + (2 * 3,14 * R * H);
6.      Litro ← Área/3;
7.      Qtde ← Litro/5;
8.      C ← Qtde * 50,00;
9.      escreva (C, Qtde);
10. fim.
```

c)
Construa um algoritmo para calcular as raízes de uma equação do 2º grau ($Ax^2 + Bx + C$), sendo que os valores A, B, C são fornecidos pelo usuário (considere que a equação possui duas raízes reais).

```
1.  início
2.      //Declaração de variáveis
3.      real: A, B, C,    //Coeficientes da equação
4.            D,          //delta
5.            X1, X2;     //raízes
6.
7.      //entrada de dados
8.      leia (A,B,C);
9.
10.     //processamento de dados
11.     D ← pot (B,2) – 4*A*C;
12.     X1 ← (-B + rad(D))/(2*A);
13.     X2 ← (-B – rad(D))/(2*A);
14.
15.     //saída de dados
16.     escreva (“Primeira raiz = ”, X1);
17.     escreva (“Segunda raiz = ”, X2);
18. fim.
```

d)
Construa um algoritmo que, tendo como dados de entrada dois pontos quaisquer do plano, P(x_1,y_1) e Q(x_2, y_2), imprima a distância entre eles.

A fórmula que efetua tal cálculo é: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$, que reescrita utilizando os operadores matemáticos adotados fica: d=rad(pot(x2-x1,2) + pot(y2-y1,2))

```
1.  início
2.        //Declaração de variáveis
3.        real: D; //Distancia calculada
4.        inteiro: X1, X2, Y1, Y2; //pontos
5.
6.        //entrada de dados
7.        leia (X1, X2, Y1, Y2); //Valores dos pontos
8.
9.        //processamento de dados
10.       D ← rad (pot(X2-X1,2) + pot(Y2-Y1,2));
11.
12.        //saída de dados
13.       escreva (“Distancia = ”, D);
14. fim.
```

e)
Faça um algoritmo para calcular o volume de uma esfera de raio R, em que R é um dado fornecido pelo usuário. O volume de uma esfera é dado por $V = \frac{4}{3}\pi R^3$.

```
1.  início
2.        //Declaração de variáveis
3.        real: R, //raio
4.              V; //volume
5.
6.        //entrada de dados
7.        leia (R);
8.
9.        //processamento de dados
10.       V ← 4/3 * 3.1416 * pot (R,3);
11.
12.        //saída de dados
13.       escreva (“Volume = ”, V);
14. fim.
```

Estruturas de Seleção

Uma estrutura de seleção permite a escolha de um grupo de ações (bloco) a ser executado quando determinadas condições, representadas por expressões lógicas ou relacionais são ou não satisfeitas.

Seleção simples: quando precisamos testar uma certa condição antes de executar uma ação.
Exemplo:

```
inicio
      //declaração de variáveis
      real: N1, N2, N3, N4, MA; // notas bimestrais e média anual
      leia (N1,N2,N3,N4); // entrada de dados
      MA ← (N1 + N2 + N3 + N4) / 4; // processamento
      escreva (MA);    // saída de dados
      se (MA >= 7)
            então
                  escreva (“Aluno Aprovado!”);
      fimse;
fim.
```

Seleção composta: quando tivermos situações em que duas alternativas dependem de uma mesma condição, uma condição ser verdadeira e outra de a condição ser falsa.
Exemplo:

```
inicio
```

```
        //declaração de variáveis
        real: N1, N2, N3, N4, MA; // notas bimestrais e média anual
        leia (N1,N2,N3,N4); // entrada de dados
        MA ← (N1 + N2 + N3 + N4) / 4; // processamento
        escreva (MA);     // saída de dados
        se (MA >= 7)
                então
                    inicio // Bloco verdade
                            escreva (“Aluno Aprovado!”);
                            escreva (“Parabéns!”);
                    fim;
                senão
                    inicio // Bloco falsidade
                            escreva (“Aluno Reprovado!”);
                            escreva (“Estude Mais!”);
                    fim;
        fimse;
fim.
```

Seleção encadeada: quando, devido à necessidade de processamento, agrupamos várias seleções.
Exemplo:
Dados três valores A, B, C, verificar se eles podem ser os comprimentos dos lados de um triângulo, se forem, verificar se compõem um triângulo eqüilátero, isósceles ou escaleno. Informar se não compuserem nenhum triângulo.
Dados de entrada: três lados de um suposto triângulo (A, B, C).
Dados de saída – mensagens: não compõem triângulo, triângulo eqüilátero, triângulo isósceles, triângulo escaleno.
O que é triângulo?
R: Figura geométrica fechada de três lados, em que cada um é menor que a soma dos outros dois.
O que é um triângulo eqüilátero?
R: um triângulo com três lados iguais.
O que é um triângulo isósceles?
R: um triângulo com dois lados iguais.
O que é um triângulo escaleno?
R: um triângulo com todos os lados diferentes.

É triângulo?	**É eqüilátero?**	**É isósceles?**	**É escaleno**	**Ações**
V	V	-	-	“Eqüilátero”
V	F	V	-	“Isósceles”
V	F	F	V	“Escaleno”
F	-	-	-	“Não é triângulo”

Traduzindo as condições para expressões lógicas:
É triângulo: (A < B + C) e (B< A + C) e (C < A + B)
É eqüilátero: (A = B) e (B = C)
É isósceles: (A = B) ou (A = C) ou (B = C)
É escaleno: (A <>B) e (B <> C) e (A <> C)

```
inicio
        inteiro: A, B, C;
        leia(A, B, C);
        se ((A < B + C) e (B < A + C) e (C < A + B))
            então
                se ((A = B) e (B = C))
                    então
                        escreva (“Triângulo Eqüilátero”);
                    senão
                        se ((A = B) ou (A = C) ou (B = C))
                            então
                                escreva (“Triângulo Isósceles”);
                            senão
                                escreva (“Triângulo Escaleno”);
                        fimse;
                fimse;
            senão
                escreva (“Estes valores não formam um Triângulo!”);
        fimse;
fim.
```

Exercício

Desenvolva um algoritmo que calcule as raízes de uma equação do 2º grau, na forma $Ax^2 + Bx + C$, levando em consideração a existência de raízes reais.
R:

```
inicio
        real: A, B, C, D, X1, X2; // coeficientes da equação, delta, raízes
        leia (A, B, C);
        D ← pot (B,2) – 4*A*C;
        se (D>0) // Duas raízes reais
           então
              inicio
                 X1 ← (-B + rad(D)) / (2*A);
                 X2 ← (-B – rad(D)) / (2*A);
                 escreva (“Primeira raiz = ”, X1, “e Segunda raiz = “, X2);
              fim;
           senão
              se (D = 0) // uma única raiz real
                 então
                    inicio
                       X1 ← -B / (2*A);
                       escreva (“Raiz = ”, X1);
                    fim;
                 senão
                    escreva (“As raízes são imaginárias”);
              fimse;
        fimse;
fim.
```

Estruturas de Seleção

se então se

Vamos supor que, em um dado algoritmo, um comando genérico W deva ser executado apenas quando forem satisfeitas as condições <Condição 1>, <Condição 2>, <Condição 3> e <Condição 4>, teríamos:

```
se <Condição 1>
   então se <Condição 2>
```

```
        então
                C2;
fimse;
se (X = V4)
        então
                C4;
fimse;
```

Somente um, e apenas um, comando pode ser executado, isto é, trata-se de uma situação excludente (se X é igual a V3, não é igual a V1 nem a V2 nem a V4).
Não se trata de uma estrutura encadeada, pois as seleções não serão interligadas. Por isso, todas as condições (X = Vn) serão avaliadas e ocorrerão testes desnecessários. Para diminuir a quantidade de testes dessa estrutura podemos transforma-la em um conjunto de seleções encadeadas, conforme o seguinte modelo:

```
se (X = V1)
   então   C1;
   senão se (X = V2)
            então   C2;
            senão se (X = V3)
                     então   C2;
                     senão se (X = V4)
                              então  C4;
                           fimse;
                  fimse;
         fimse;
fimse;
```

Seleção de múltipla escolha
Quando um conjunto de valores discretos precisa ser testado e ações diferentes são associadas a esses valores, estamos diante de uma seleção encadeada homogênea do tipo se senão se. Como essa situação é bastante freqüente na construção de algoritmos que dependem de alternativas, utilizaremos uma estrutura específica para estes casos, a seleção de múltipla escolha.
O modelo genérico desse tipo de repetição é o seguinte:

```
escolha X
        caso V1: C1;
        caso V2: C2;
        .
        .
        .
        caso Vn: Cn;
        caso contrário: Cn+1;
fimescolha;
```

Exemplo:
Construa um algoritmo que, tendo como dados de entrada o preço de um produto e seu código de origem, mostre o preço junto de sua procedência. Caso o código não seja nenhum dos especificados, o produto deve ser encarado como importado. Siga a tabela de códigos a seguir:

Código de origem	**Procedência**
1	Sul
2	Norte
3	Leste
4	Oeste
5 ou 6	Oeste
7,8 ou 9	Sudeste
10 até 20	Centro-Oeste
25 até 30	Nordeste

```
inicio
        //declaração de variáveis
        real: Preco;
        inteiro: Origem;
        leia (Preço, Origem); // entrada de dados
        escolha (Origem)
                caso 1: escreva(Preço, " – produto do Sul");
```

```
            caso 2: escreva(Preço, " – produto do Norte");
            caso 3: escreva(Preço, " – produto do Leste");
            caso 4: escreva(Preço, " – produto do Oeste");
            caso 7, 8, 9: escreva(Preço, " – produto do Sudeste");
            caso 10..20: escreva(Preço, " – produto do Centro-Oeste");
            caso 5, 6, 25..30: escreva(Preço, " – produto do Nordeste");
            caso contrario: escreva(Preço, " – produto Importado");
      fimescolha;
fim.
```

Exercício

Elabore um algoritmo que calcule o que deve ser pago por um produto, considerando o preço normal de etiqueta e a escolha da condição de pagamento. Utilize os códigos da tabela, a seguir, para ler qual a condição de pagamento escolhida e efetuar o cálculo adequado.

Código	Condição de pagamento
1	À vista em dinheiro ou cheque, recebe 10% de desconto.
2	À vista no cartão de crédito, recebe 5% de desconto.
3	Em duas vezes, preço normal de etiqueta sem juros.
4	Em três vezes, preço normal de etiqueta mais juros de 10%.

R:

```
inicio
      //declaração de variáveis
      real: P, NP; // preço do produto, novo preço conforme a condição escolhida
      inteiro: COD; // código do produto
      leia (P, COD); // entrada de dados
      escolha (COD)
            caso 1:
               inicio
                  NP ← P * 0.90; // desconto de 10
                  escreva("Preço à vista com desconto = ", NP);
               fim;
            caso 2:
               inicio
                  NP ← P * 0.95; // desconto de 5
                  escreva("Preço no cartão com desconto = ", NP);
               fim;
            caso 3:
               inicio
                  NP ← P / 2; // duas vezes sem acrescimo
                  escreva("Duas parcelas de = ", NP);
               fim;
            caso 4:
               inicio
                  NP ← (P * 1.10)/3; // acréscimo de 10
                  escreva("Três parcelas de = ", NP);
               fim;
            caso contrario: escreva("Código inexistente!");
      fimescolha;
fim.
```

Estruturas de Repetição

Repetição com teste no início

Consiste em uma estrutura de controle de fluxo de execução que permite repetir diversas vezes um mesmo trecho do algoritmo, porém, sempre verificando antes de cada execução se é permitido executar o mesmo trecho.

O modelo genérico desse tipo de repetição é o seguinte:

```
enquanto <condição> faça
      C1;
      C2;
      .
      .
      .
      Cn;
fimenquanto;
```

Quando o resultado de <condição> for falso, o comando de repetição é abandonado. Se já a primeira vez o resultado é falso, os comandos não são executados nenhuma vez.
Por exemplo, para inserir o cálculo da média dos alunos em um laço de repetição – utilizamos a estrutura enquanto – que <condição> utilizaríamos?
A condição seria que a quantidade de médias calculadas fosse menor ou igual a 50; porém, o que indica quantas vezes a média foi calculada? A estrutura (enquanto) não oferece esse recurso; portanto, devemos estabelecer um modo de contagem, o que pode ser feito com a ajuda de um contador representado por uma variável com um dado valor inicial, o qual é incrementado a cada repetição.

Exemplo (Contador):
```
inteiro: CON; // declaração do contador
CON ← 0; // inicialização do contador
CON ← CON + 1; // incrementar o contador de 1
```

Exemplo: Algoritmo – Média aritmética para 50 alunos

```
inicio
        //declaração de variáveis
        real: N1, N2, N3, N4, MA; // notas bimestrais e média anual
        inteiro: CON; // contador
        CON ← 0; // inicialização do contador
        enquanto (CON < 50) faça   // teste da condição de parada
           leia (N1,N2,N3,N4); // entrada de dados
           MA ← (N1 + N2 + N3 + N4) / 4; // processamento
           escreva (MA);// saída de dados
           se (MA >= 7)
                então
                   inicio // Bloco verdade
                        escreva (“Aluno Aprovado!”);
                        escreva (“Parabéns!”);
                   fim;
                senão
                   inicio // Bloco falsidade
                        escreva (“Aluno Reprovado!”);
                        escreva (“Estude Mais!”);
                   fim;
           fimse;
           CON ← CON + 1; //incrementar o contador em um
        fimenquanto;
fim.
```

Em uma variação do algoritmo acima, poderíamos calcular a média geral da turma, que seria a média aritmética das 50 médias anuais: (M1 + M2 + M3 + . . . + M49 + M50) / 50.
Podemos utilizar nessa situação as vantagens da estrutura de repetição, fazendo um laço que a cada execução acumule em uma variável, conhecida como acumulador, o somatório das médias anuais de cada aluno.

Exemplo (Acumulador):
```
inteiro: ACM, X; //declaração do acumulador, variável qualquer
ACM ← 0; // inicialização do acumulador
ACM ← ACM + X; // acumular em ACM o valor anterior mais o valor de X
```

Exemplo: Algoritmo – Média aritmética de 50 alunos

```
inicio
        //declaração de variáveis
        real: MA, ACM, MAT; // media anual de um dado aluno, acumulador, média anual da turma
        inteiro: CON; // contador
        CON ← 0; // inicialização do contador
        ACM ← 0; // inicialização do acumulador
        enquanto (CON < 50) faça   // teste da condição de parada
           leia (MA); // entrada de dados
           ACM ← ACM + MA; // processamento
           CON ← CON + 1; //incrementar o contador em um
        fimenquanto;
        MAT ← ACM/50;  // Cálculo da média anual da turma
```

```
    escreva (“Média anual da turma = ”, MAT);
fim.
```

O algoritmo construído acima utiliza o pré-conhecimento da quantidade de alunos da turma da qual se desejava a média geral, o que permitiu construir um laço de repetição com quantidade pré-determinada de execuções. Entretanto, se não soubéssemos quantos eram os alunos, o que faríamos para controlar o laço de repetição? Precisaríamos de um laço que fosse executado por uma quantidade indeterminada de vezes. Assim, teríamos de encontrar outro critério de parada, que possibilitasse que o laço fosse finalizado após a última média anual (independente de quantas sejam) ter sido informada. Isso pode ser feito utilizando um valor predefinido como finalizador, a ser informado após a última média.
Para aplicar tal conceito ao algoritmo da média geral da turma, usaremos como finalizador o valor -1 que, quando encontrado, encerra o laço sem ter seu valor computado ao acumulador.

Exemplo:

```
inicio
    //declaração de variáveis
    real: MA, ACM, MAT; // media anual de um dado aluno, acumulador, média anual da turma
    inteiro: CON; // contador
    CON ← 0; // inicialização do contador
    ACM ← 0; // inicialização do acumulador
    MA ← 0; // inicialização da variável de leitura
    enquanto (MA <> -1) faça   // teste da condição de parada
        leia (MA); // entrada de dados
        se (MA <> -1) então // evita acumulação do finalizador
            inicio
                ACM ← ACM + MA; // processamento
                CON ← CON + 1; //incrementar o contador em um
            fim;
        fimse;
    fimenquanto;
    se (CON > 0)  // houve pelo menos uma execução
        então
            inicio
                MAT ← ACM/CON;  // Cálculo da média anual da turma
                escreva (“Média anual da turma = ”, MAT);
            fim;
        senão
                escreva (“Nenhuma média válida fornecida”);
    fimse;
fim.
```

Repetição com teste no final

Para realizar a repetição com teste até no final, utilizaremos a estrutura repita, que permite que um bloco ou ação primitiva seja repetido até que uma determinada condição seja verdadeira. O modelo genérico desse tipo de repetição é o seguinte:

```
repita
    C1;
    C2;
    .
    .
    .
    Cn;
até <condição>;
```

Exemplos:

a) Média anual de 50 alunos e calcula a média geral da turma.
Algoritmo – média com repita

```
inicio
    //declaração de variáveis
    real: MA, ACM, MAT; // media anual de um dado aluno, acumulador, média anual da turma
    inteiro: CON; // contador
    CON ← 0; // inicialização do contador
    ACM ← 0; // inicialização do acumulador
    repita
```

```
            leia (MA); // entrada de dados
            ACM ← ACM + MA; // processamento
            CON ← CON + 1; //incrementar o contador em um
        até (CON >= 50);  // teste da condição de parada
        MAT ← ACM/50;  // Cálculo da média anual da turma
        escreva (“Média anual da turma = ”, MAT);
fim.
```

b) Imagine uma brincadeira entre dois colegas, na qual um pensa um número e o outro deve fazer chutes até acertar o número imaginado. Como dica, a cada tentativa é dito se o chute foi alto ou foi baixo. Elabore um algoritmo dentro deste contexto que leia o número imaginado e os chutes, ao final mostre quantas tentativas foram necessárias para descobrir o número.

Algoritmo – descoberta do número

```
inicio
   inteiro: NUM, CHUTE, TENT;   // numero inicial a ser descoberto, chute, quantidade de tentativas
   TENT ← 0;
   leia(NUM);
   repita
      leia(CHUTE);
      TENT ← TENT + 1;
      se (CHUTE > NUM)
         então escreva (“Chutou alto!”);
         senão se (CHUTE < NUM)
                  então escreva (“Chutou baixo!”);
               fimse;
      fimse;
   até (NUM = CHUTE);
   escreva (TENT);
fim.
```

Repetição com variável de controle
Nas estruturas de repetição vistas até agora, ocorrem casos em que se torna difícil determinar o número de vezes em que o bloco será executado. Sabemos que ele será executado enquanto uma condição for satisfeita – *enquanto* – ou até que uma condição seja satisfeita – *repita*. A estrutura *para* é diferente, já que sempre repete a execução do bloco um número predeterminado de vezes, pois ela não prevê uma condição e possui limites fixos. O modelo genérico para a estrutura de repetição *para* é o seguinte:

```
para V de vi até vf passo p faça
        C1;
        C2;
        .
        .
        .
        Cn;
fimpara;
```

Em que:
- V é a variável de controle;
- vi é o valor inicial da variável V;
- vf é o valor final da variável V, ou seja, o valor até o qual ela vai chegar;
- p é o valor do incremento dado à variável V.

Exemplos:
a) Média anual de 50 alunos (com para)

```
inicio
        //declaração de variáveis
        real: MA, ACM, MAT; // media anual de um dado aluno, acumulador, média anual da turma
        inteiro: V; // variável de controle
        ACM ← 0; // inicialização do acumulador
        para V de 1 até 50 passo 1 faça
            leia (MA); // entrada de dados
            ACM ← ACM + MA; // processamento
        fimpara;
        MAT ← ACM/50;  // Cálculo da média anual da turma
```

```
        escreva (“Média anual da turma = ”, MAT);
fim.
```

b) Elabore um algoritmo que efetue a soma de todos os números ímpares que são múltiplos de 3 e que se encontram no conjunto dos números de 1 até 500.

```
inicio
        //declaração de variáveis
        inteiro: SI, V; // Soma dos numero impares múltiplos de três, variável de controle
        SI ← 0;
        para V de 1 até 500 passo 1 faça
            se (V mod 2 = 1) // número é ímpar?
                então
                    inicio
                        se (V mod 3 = 0) // múltiplo de três?
                            então
                                SI ← SI + V;
                        fimse;
                    fim;
            fimse;
        fimpara;
        escreva (“Soma = ”, SI);
fim.
```

c) Elabore um algoritmo que simule uma contagem regressiva de 10 minutos, ou seja, mostre 10:00, e então 9:59, 9:58, ..., 9:00; 8:59, 8:58, até 0:00

Algoritmo – Contagem regressiva

```
inicio
        //declaração de variáveis
        inteiro: MIN, SEG; // contador dos minutos, contador dos segundos
        escreva (“10:00”);
        para MIN de 9 até 0 passo -1 faça
            para SEG de 59 até 0 passo -1 faça
                escreva(MIN, “:”, SEG);
            fimpara;
        fimpara;
fim.
```

Exercícios:

a) Elabore um algoritmo que, utilizando as três estruturas de repetição, imprima a tabuada do número 5:

Algoritmo – Tabuada do número 5 usando enquanto

```
inicio
    inteiro: CON;
    CON ← 1;
    enquanto (CON <= 10) faça
        escreva (CON, “ x 5 = ”, CON * 5);
        CON ← CON +1;
    fimenquanto;
fim.
```

Algoritmo – Tabuada do número 5 usando repita

```
inicio
    inteiro: CON;
    CON ← 1;
    repita
        escreva (CON, “ x 5 = ”, CON * 5);
        CON ← CON +1;
    até (CON > 10);
fim.
```

Algoritmo – Tabuada do número 5 usando para

```
inicio
    inteiro: CON;
    CON ← 1;
    para CON de 1 até 10 passo 1 faça
        escreva (CON, " x 5 = ", CON * 5);
    fimpara;
fim.
```

b) **Modifique o algoritmo para que ele imprima a tabuada de quaisquer números, sendo que esses são fornecidos pelo usuário, até encontrar como finalizador -1. Sabendo que o primeiro número-base fornecido não é -1:**

Algoritmo – Tabuada de qualquer número usando enquanto

```
inicio
    inteiro: N, CON;
    leia (N);
    enquanto (N <> -1) faça
        CON ← 1;
            enquanto (CON <= 10) faça
                escreva (CON, " x ", N, " = ", CON * N);
                CON ← CON +1;
            fimenquanto;
            leia (N);
    fimenquanto;
fim.
```

Algoritmo – Tabuada de qualquer número usando repita

```
inicio
    inteiro: N, CON;
    leia (N);
    repita
        CON ← 1;
             repita
                escreva (CON, " x ", N, " = ", CON * N);
            CON ← CON +1;
        até (CON > 10);
            leia (N);
    até (N = -1);
fim.
```

Algoritmo – Tabuada de qualquer número usando para

```
inicio
    inteiro: N, COM, X;
    leia (N);
    para X de 1 até ? para 1 faça // número de repetições é indefinido
        CON ← 1;
            para CON de 1 até 10 passo 1 faça
                escreva (CON, " x ", N, " = ", CON * N);
            fimpara;
            leia (N);
    fimapara;
fim.
```

Obs: Verificamos nesse algoritmo a impossibilidade de construir esse algoritmo utilizando a estrutura para, pois esta exige que o número de repetições, além de ser finito, seja predeterminado.